Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Distilling the Knowledge of SlovakBERT

Diploma Thesis

2022
Bc. Ivan Agarský

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Distilling the Knowledge of SlovakBERT

Diploma Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	prof. Ing. Igor Farkaš, Dr.
Consultant:	Mgr. Marek Šuppa

Bratislava, 2022
Bc. Ivan Agarský

76214648

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Ivan Agarsky
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Distilling the Knowledge of SlovakBERT
Destilácia znalostí modelu SlovakBERT

Anotácia: Veľké predtrénované jazykové modely sa stali štandardným základným prvkom pre modely riešiace rôzne úlohy spracovania prirodzeného jazyka. V posledných rokoch sa značná časť výskumu venovala ich prispôsobeniu pre viacjazyčné prostredie, ako aj pre konkrétne jazyky, ako napríklad SlovakBERT pre slovenčinu. Tieto modely sú však vo všeobecnosti veľké, čo ich robí neuplatniteľnými pre praktické aplikácie. Jedným z prístupov znižovania veľkosti modelu je destilácia znalostí, ktorá priniesla priaznivé výsledky pre anglické modely, ale nebola extenzívne testovaná pre iné jazyky. Cieľom tejto práce je preskúmať jej aplikáciu na model SlovakBERT so zameraním na zmenšenie jeho veľkosti a minimálnym dopadom na jeho výkon.

Cieľ: 1) Preskúmajte súčasný stav v oblasti destilácie znalostí veľkých predtrénovaných jazykových modelov. 2) Navrhnite, implementujte a vyhodnoťte prístup k destilácii znalostí pre model SlovakBERT.

Literatúra: Pikuliak, Matúš, et al. SlovakBERT: Slovak Masked Language Model. arXiv:2109.15254 (2021).
Sanh, Victor, et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." arXiv:1910.01108 (2019).
Goldberg, Y. (2017). Neural network methods for natural language processing. Synthesis lectures on human language technologies, 10(1), 1-309.

Vedúci:	prof. Ing. Igor Farkaš, Dr.
Konzultant:	Mgr. Marek Šuppa
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.
Dátum zadania:	16.09.2021

Dátum schválenia: 21.09.2021

prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

....
študent

....
vedúci práce

76214648

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Ivan Agarsky
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Distilling the Knowledge of SlovakBERT

Annotation: Large pre-trained language models have become a standard building block for models tackling various Natural Language Processing (NLP) tasks. In the recent years, a significant body of research has been devoted to adapting them to the multilingual setting, as well as for specific languages, such as for instance the SlovakBERT model for the Slovak language. These models, however, are generally large in size, rendering them unfeasible for practical applications. One of the approaches of decreasing the model size is knowledge distillation, which has yielded favorable results for English models but has not been extensively tested for other languages. This thesis aims to investigate its application on the SlovakBERT model, with the intent to decrease its size while minimally affecting its performance.

Aim: 1) Review the current state of the art of knowledge distillation of large pre-trained language models. 2) Propose, implement and evaluate a knowledge distillation approach for the SlovakBERT model.

Literature: Pikuliak, Matúš, et al. SlovakBERT: Slovak Masked Language Model. arXiv:2109.15254 (2021).
Sanh, Victor, et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." arXiv:1910.01108 (2019).
Goldberg, Y. (2017). Neural network methods for natural language processing. Synthesis lectures on human language technologies, 10(1), 1-309.

Supervisor:	prof. Ing. Igor Farkaš, Dr.
Consultant:	Mgr. Marek Šuppa
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	16.09.2021
Approved:	21.09.2021

prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

....
Student

....
Supervisor

Acknowledgements: I wish to express my gratitude to my consultant Marek for his valuable help and time devoted to guiding me through the makings of this thesis. My thanks also go to my supervisor prof. Farkaš for his helpful remarks and text reviews.

Abstrakt

Hlboké neurónové siete sa stali úspešným prístupom k modelovaniu prirodzeného jazyka. Presnosť modelov má tendenciu rásť s veľkosťou siete. Jazykové modely stále vyžadujú viac a viac pamäte na trénovanie a používanie. Do centra pozornosti výskumnej komunity sa dostala technika na zmenšenie veľkosti neurónovej siete pri zachovaní takmer celého jej výkonu, nazývaná znalostná destilácia.

V práci sa zaoberáme jazykovo špecifickou destiláciu znalostí a ukazujeme, že je to životaschopná technika na zníženie veľkosti modelu pri zachovaní takmer celej jeho presnosti. Destilované modely hodnotíme na štyroch úlohách na porozumenie jazyka, z ktorých niektoré sú strojovo preložené do slovenčiny, a to STS a BoolQ. Okrem toho sme ukázali, že spriemerňovanie logitov a skrytých stavov pri vykonávaní destilácie vedomostí od viacerých učiteľov, ktorí videli rovnaký súbor údajov o školení, neposkytuje výhodu študentskému modelu. Naše destilované modely dosahujú 91% až 99% presnosti pôvodného modelu, pričom majú o 46% menej parametrov.

Kľúčové slová: destilácia znalostí, BERT, slovenčina

Abstract

Deep neural networks have become a successful approach to natural language modeling. Model accuracy tends to increase with network size. The language models require more and more memory to train and use. A focus on reducing the size of the neural network while maintaining almost all of its performance, called knowledge distillation, has come to the forefront of the research community.

In this work we deal with language-specific knowledge distillation and show that it is a viable technique for reducing the size of the model while maintaining almost all its accuracy. We evaluate distilled models on four language understanding tasks, some of which are machine-translated into Slovak, namely STS and BoolQ. In addition, we show that averaging logits and hidden states when performing knowledge distillation from multiple teachers, who have seen the same set of training data, does not provide an advantage to the student model. Our distilled models achieve from 91% to 99% accuracy of the original model, but have 46% fewer parameters.

Keywords: knowledge distillation, BERT, Slovak language

Contents

Chapter 1

Introduction

Supervised deep neural network training in most cases requires a significantly large dataset. Collecting or generating such datasets for specific Natural Language Processing (NLP) tasks can be expensive and non-practical. Problems with creating large enough datasets made researchers find ways for improving language model performance, besides increasing dataset size. Unsupervised pre-training of large language models demonstrated a significant increase in performance across all NLP tasks and presented itself as a viable alternative to the need for creating large datasets for every specific task. The process of adapting a pre-trained language model to a specific task is called *fine-tuning*. These specific tasks are also called *downstream* tasks. Large pre-trained language models like BERT and GPT caused a rise in interest in natural language processing and understanding (Brown et al., 2020; Devlin et al., 2019). Lately, researchers devoted significant time to creating multilingual models, as well as models for specific non-English languages, such as SlovakBERT (Pikuliak et al., 2021). It has been shown that increasing the model size and adding additional layers positively influences model performance across all downstream tasks. This creates problems with model practicality, especially on mobile and edge devices. The most recent largest models are unusable even on high-end PCs, due to a large number of parameters and long inference time.

Recently, knowledge distillation was shown to be a viable approach to decreasing model size while retaining nearly all of its performance (Gou et al., 2021; Sanh et al., 2020). Decreasing the model size automatically decreases its inference time, too. Knowledge distillation yielded favourable results for English models but has not been extensively tested for other languages. We aim to apply this method to the SlovakBERT model and experiment with various modifications to the distillation setup, including distillation with multiple teachers. Additionally, we evaluate the distilled models on four downstream tasks which to some extent require language understanding. Two of the tasks are fine-tuned on a machine-translated dataset.

This thesis is split into four main parts. Chapter *Preliminaries* introduces language modelling, briefly describes neural networks and presents an established text vector representation called word embedding. Chapter *Related work* presents an overview of modern language modelling using deep neural networks, knowledge distillation and evaluation of language models. Chapter *Proposed solution* describes in detail the conducted experiments and datasets used. Finally, chapter *Results* presents and compares the results of the conducted experiments. Some ideas for further work can be found in the *Conclusion* chapter.

Chapter 2

Preliminaries

For the purposes of this thesis, any human language can be considered a natural language, excluding programming languages. Constructed languages, such as Esperanto (Korzhenkov, 2010), can also be considered natural languages for the purposes of natural language processing.

2.1 Language model

A language model is constructed by assigning probabilities to either every sentence or every sequence of tokens. Tokens can be words, characters or bytes. The language model also assigns a probability for the likelihood of a given word following a sequence of words. If we denote a sentence with $\boldsymbol{w}$ then its probability is the probability of all words w_i in that sentence appearing in that order:

$$P(\boldsymbol{w}) = P(w_1, w_2, \ldots, w_k) \tag{2.1}$$

The most straightforward way to make a language model for any natural language would be to assign probabilities to sentences based on the number of occurrences in the whole available corpus:

$$P(\boldsymbol{w}) = \frac{\#(\boldsymbol{w})}{S} \tag{2.2}$$

where S is the number of sentences in the whole corpus. With infinite data, this language model would be "correct" — but in practice, this model needs to calculate the probabilities for every possible sentence, while drawing from a limited corpus. To make language models more practical, ***n*-gram models** introduce a simplification. Instead of looking at the whole sequence when assigning likelihood for the next word (or a sequence of words) they consider only the last $n-1$ words :

$$P(w_k \mid w_{k-1}, \ldots, w_1) \approx P(w_k \mid w_{k-1}, \ldots, w_{k-n+1}) \tag{2.3}$$

A bigram (2-gram) language model would calculate the probability of a three-token sequence in the following way:

$$P(w_1, w_2, w_3) = P(w_1 \mid START) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_2) \cdot P(END \mid w_3) \tag{2.4}$$

With a small n, the model will fail to capture longer sentence-spanning contexts. Large n will perform better, but will also demand exponentially more data.

N-gram models are used to predict the next word in a sequence. A straightforward way to assign a probability to the last token in a sequence is to count the number of sequences in the corpus containing the same beginning with and without that specific token at the end and divide them:

$$P(w_3 \mid w_1, w_2) = \frac{P(w_1, w_2, w_3)}{P(w_1, w_2)} \tag{2.5}$$

Finally, we choose the token with the highest probability. This simple method can fail when the denominator is zero. We can alleviate this problem by introducing smoothing to the probability distribution. The simplest way is to add one to the number of occurrences of every possible combination of tokens. This smoothing is called *Laplace smoothing*.

2.2 Word embedding

In order to better represent textual data inside a computer, a common strategy is to assign vector representation to each word. A word-to-vector mapping is called a *word embedding*. The simplest maps are one-hot vectors which for i^{th} word in the vocabulary contain one in i^{th} dimension and zeros otherwise. It is easily seen that the distances between all words in the dictionary produced by this word-to-vector map are equal to $\sqrt{2}$. This is problematic because we want to be able to detect similar words by looking at their vectors. One way to check if vectors of semantically similar words are mapped closer to each other than dissimilar words is to calculate cosine similarity as shown below:

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} \tag{2.6}$$

where $\|\boldsymbol{x}\|$ represents the Euclidean norm of the real-valued vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.

The following Python code snippet demonstrates cosine similarity for embedded vectors[1]:

```python
print(king.similarity(banana))  # 0.2175
print(king.similarity(man))  # 0.4088
```

[1]Using *spacy* library (Honnibal & Montani, 2017)

```
print(king.similarity(emperor))  # 0.5809
print(king.similarity(king))  # 1.0
```

Arithmetic operations can be applied to embedded vectors, to some extent, as can be seen in the following example:

```
v = king.vector - man.vector + woman.vector
torch_v = torch.FloatTensor(v)
torch_queen = torch.FloatTensor(queen.vector)
result = cosine_similarity(torch_v, torch_queen, dim=0)
print(result)  # 0.7881
```

where the vector "$king - man + woman$" is semantically similar to the vector "*queen*". The example was taken from (Mikolov, Yih, et al., 2013). On the other hand, a similar example using the words *bookkeeper*, *book* and *keeper* does not produce the same result.

```
v = bookkeeper.vector - book.vector
torch_v = torch.FloatTensor(v)
torch_keeper = torch.FloatTensor(keeper.vector)
result = cosine_similarity(torch_v, torch_keeper, dim=0)
print(result)  # 0.0597
```

Although this may be an issue with the library used, there are limitations to applying arithmetic operations to embedded vectors due to some words having multiple meanings.

One of the earliest methods for constructing word embedding was counting word co-occurrences (Lund & Burgess, 1996). A matrix $\boldsymbol{N}$ is constructed such that $\boldsymbol{N_{ij}}$ equals to number of times word w_i appears near w_j in a L-sized window. Word vector is then simply the row/column of the matrix with the appropriate index. Early topic models, such as LSA (Deerwester et al., 1990), were assigning vectors to whole documents, which were then compared using cosine similarity to measure the similarity of their contents.

2.2.1 Word2Vec

Word embeddings, such as Word2Vec, construct real-valued and high-dimensional vectors of semantically similar words that are close together in vector space (Mikolov, Chen, et al., 2013; Mikolov, Sutskever, et al., 2013). Word2Vec achieves this by encoding information about the context of the word inside its vector representation. The context of a word is the words appearing in a fixed-size window around the target word. Word2Vec algorithm has two variants: Skip-grams (SG) and Continous Bag of Words (CBOW). SG is an iterative algorithm which can predict the context of a word. It builds a probability distribution over words in the vocabulary. It can be summarised as follows:

1. slide a fixed-size window centred at every word in the text corpus
2. compute probabilities of context words for the current central word
3. adjust the word vectors to increase these probabilities

More specifically, we maximize the following function:

$$J'(\boldsymbol{\theta}) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} \mid w_t; \boldsymbol{\theta}) \tag{2.7}$$

where T is the corpus, m is the sliding window and $\boldsymbol{\theta}$ are the parameters of the model. i.e. the word vectors. This objective function is more commonly rewritten as the average negative log-likelihood of the original function:

$$J(\boldsymbol{\theta}) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m} \log P(w_{t+j} \mid w_t; \boldsymbol{\theta}) \tag{2.8}$$

Probabilities are computed by applying the softmax function on the dot-product of word vectors of central and context words. The Word2Vec model usually has two vectors for every word, one when the word is in the centre and one when it is outside the centre. Therefore the probability calculation looks as follows:

$$P(w_{t+j} \mid w_t) = \frac{\exp(\boldsymbol{u}_{w_{t+j}}^T \cdot \boldsymbol{v}_{w_t})}{\sum_{z=1}^{v} \exp(\boldsymbol{u}_z^T \cdot \boldsymbol{v}_{w_t})} \tag{2.9}$$

The objective function is optimized by using gradient descent. When the optimization finishes, we usually throw away the context vectors and use only central word vectors. This allows us to arrive at word vectors which fulfill the similarity requirements for word embedding. A modification of SG that causes much faster training is called *negative sampling*. It does not update every context word vector in each step, but only a randomly chosen subset of a fixed size. The CBOW variant of Word2Vec is similar, albeit it optimizes for predicting the central word based on context words.

2.2.2 GloVe

Skip-gram model contains an inherent downside. It trains only on local context windows, therefore poorly utilizes the statistics of the corpus and global co-occurrence counts. A new model, called Global Vectors (GloVE), was proposed to fix this downside (Pennington et al., 2014). This model combines prediction methods, such as Word2Vec, with count-based methods, such as simple co-occurrence counts. The authors propose optimizing the following function:

$$J(\boldsymbol{\theta}) = \sum_{w_i, w_j \in V} f(\boldsymbol{N}_{ij}) \cdot (\boldsymbol{u}_{w_j}^T \cdot \boldsymbol{v}_{w_i} + b_{w_j} + b_{w_i} - \log \boldsymbol{N}_{ij})^2 \tag{2.10}$$

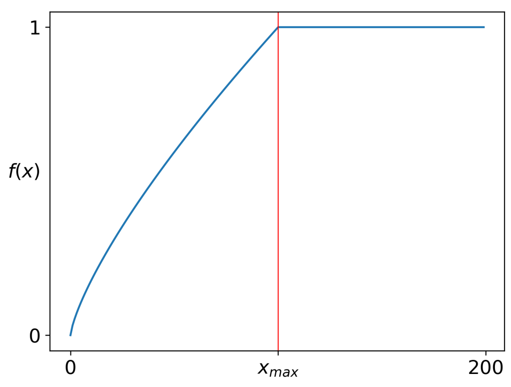


Figure 2.1: The weighting function used in GloVE model.

where V is the vocabulary, $\boldsymbol{N}$ is the co-occurrence matrix, b_{w_i} and b_{w_j} are bias terms, and f is a weighting function that penalizes rare events and prevents frequent events from over-weighting. The weighting function is defined in parts as follows:

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^{\alpha} & \text{if } x > x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{2.11}$$

where $x_{\max}$ and α are hyper-parameters usually set to 100 and 0.75, respectively. The weighting function with the most common hyper-parameters is shown in Figure 2.1.

2.3 Neural networks

A neural network is a powerful model which can approximate any real continuous function. Suppose a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ and the desired accuracy $\epsilon > 0$ is given, then we are guaranteed, by the universal approximation theorem (Hornik et al., 1989), that there exists a neural network whose output $g(\boldsymbol{x})$ satisfies $|g(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon$, for all inputs $\boldsymbol{x}$.

The simplest neural network is the single-layer perceptron, which contains just the input and the output neurons. Single-layer perceptron was shown to be incapable of learning some functions, such as XOR (Minsky & Papert, 1969). On the other hand, a perceptron with a hidden layer, which is a layer between the input and the output layer, could approximate any function f from the universal approximation theorem to arbitrary precision, albeit with a potentially very large number of neurons in the hidden layer. The number of layers in a neural network is considered to be its depth. The number of neurons in a layer is considered to be the layer width.

The most common type of neural network is the fully-connected feed-forward neural network. The fully-connected network consists of fully-connected layers, which contain connections between each neuron from the previous layer and the neurons from the

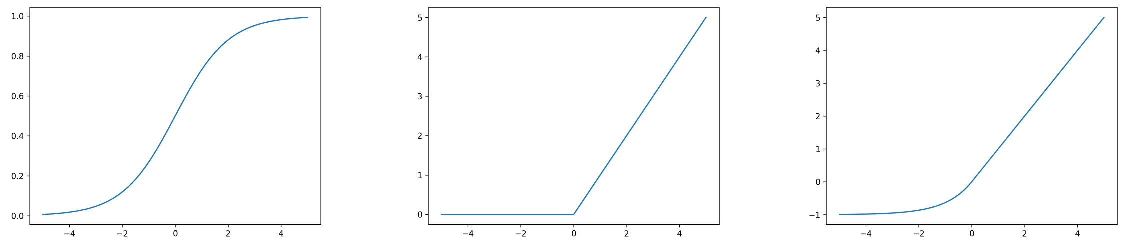


Figure 2.2: Common activation functions. From left to right: The sigmoid, The Rectified Linear Unit (ReLU), The Exponential Linear Unit (ELU).

current layer. Feed-forward networks do not contain a cycle in the graph that the network connections make. Each neuron contains an activation function, which is applied to the combined inputs entering the neuron. The fully-connected layer can be mathematically modelled with a weight matrix $\boldsymbol{W}$, a bias term b and an activation function f:

$$\boldsymbol{y} = f(\boldsymbol{W} \cdot \boldsymbol{x} + b) \tag{2.12}$$

The most common activation functions are S-shaped logistic functions:

$$f(x) = \frac{L}{1 + \exp(-k(x - x_0))} \tag{2.13}$$

where x_0 is the location of midpoint, L is the function's maximum and k is its steepness. The most common logistic function is the sigmoid with $L = 1$, $k = 1$, $x_0 = 0$. Common activation functions are shown in Figure 2.2.

Changing the weights of the neural network to better approximate a given loss function (objective) is called training of the network. Networks are usually trained through an algorithm called backpropagation (Rumelhart et al., 1986). While training the network, the loss function is continuously evaluated, its gradient is calculated and the parameters of the network are updated using the backpropagation algorithm. This operation can be written as

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.14}$$

where $\boldsymbol{\theta}$ are the network parameters, α is the learning rate and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the gradient of the loss function w.r.t. the parameters $\boldsymbol{\theta}$. In this process, the parameters $\boldsymbol{\theta}$ are moving in the opposite direction of the gradient, therefore moving closer to the local minima of the parameter space. This is called gradient descent.

Calculating the gradient for a large dataset can be very expensive. A modification of the original algorithm in which we calculate the gradient for either every example separately or a randomly drawn batch of examples is called stochastic gradient descent (SGD). This is the most common learning algorithm for neural networks. The batch size can be a very important hyperparameter for the training process. A large batch size might not fit into memory, but we can simulate a large batch size by doing gradient accumulation. Instead of updating the parameters of the model after every batch, we instead do an update every n batches. This technique can be useful in situations with limited resources.

One pass through every example in the dataset is called a training epoch. Most neural networks require multiple epochs to learn to approximate the desired function. The learning rate is usually changed during the training, depending on the epoch or closeness to the local minimum point of the parameter space. Sometimes the update of the parameters can be too large, therefore preventing us from reaching the minimum point. This can be resolved by doing gradient norm scaling, which is a method that rescales the values of the gradient if the gradient norm exceeds a given threshold.

Choosing the right loss function for the neural network is very important. Three loss functions will be mentioned in this work: cross-entropy, KL divergence and cosine embedding. Cross entropy is a loss function suitable for classification tasks. For n-class problem it is defined as:

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(s_i) \tag{2.15}$$

where $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$ is the ground truth and $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$ is the network output for the i^{th} example. Kullback–Leibler (KL) divergence measures how much is a probability distribution Q different from a reference probability distribution P:

$$\mathcal{L}_{\text{KL}}(P \mid\mid Q) = \sum_{i}^{n} (p_i \log p_i - p_i \log q_i) \tag{2.16}$$

where Q is the ground truth distribution and P is the model output distribution. This loss function is used when comparing two output token probability distributions from two language models. Cosine embedding loss measures whether two given vectors x_1 and x_2 that should be similar/dissimilar are actually similar/dissimilar. This is measured using cosine similarity:

$$L_{\text{EMB}} = \begin{cases} 1 - \cos(x_1, x_2), & \text{if they should be similar} \\ \max(0, \cos(x_1, x_2)) & \text{if they should be dissimilar} \end{cases} \tag{2.17}$$

When optimizing loss functions we can encounter highly non-convex parameter spaces that can prevent us from easily finding the minimum point. Various modifications of SGD, such as Adam (Kingma & Ba, 2014), improve its convergence speed and behaviour in saddle points and flat surfaces.

The adaptive moment estimation (Adam) is an optimization method which adaptively sets the learning rate for each parameter. It accumulates exponentially decaying average of past gradients and squared gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.18}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.19}$$

where β_1 and β_2 are the hyperparameters of the Adam optimizer usually set to 0.9 and 0.999, respectively. The final update rule is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon} \cdot \frac{m_t}{1 - \beta_1^t} \tag{2.20}$$

where α is the learning rate and ϵ is the third hyperparameter usually set a very small value, such as $1 \cdot 10^{-7}$. The Adam algorithm has become the default choice for most researchers.

Chapter 3

Related work

In this chapter, we look at language modelling using deep neural networks. We explain how the attention layer in the Transformer (Vaswani et al., 2017) works. Various BERT (Devlin et al., 2019) variants are described, including SlovakBERT (Pikuliak et al., 2021). Several approaches to knowledge distillation are presented. Finally, we touch upon the modern evaluation of language models.

3.1 Recurrent neural networks

Natural language consists of letters, words, sentences and higher-level units such as documents. Ordering of letters in a word, words in a sentence and sentences in a document is very important. Ideally, we would want a neural network to be able to remember some short-term or even long-term dependencies in text.

Early recurrent neural networks (RNNs), like simple recurrent networks (Elman, 1990), were shown to be computationally stronger than standard multilayer perceptrons because they can have a representation of a state, making them sensitive to sequential information. The output of these networks is affected by all previous inputs, in theory. In practice, training early RNNs with backpropagation proved to be computationally difficult due to vanishing or exploding gradients that occurred only in tasks with long-term dependencies. This is caused by the multiplicative nature of the backpropagation algorithm.

3.1.1 LSTM

A new RNN architecture called *Long short-term memory* (LSTM) was introduced by Hochreiter and Schmidhuber (1997) to solve the vanishing gradient problem. However, it can still suffer from the exploding gradient problem. LSTM architecture is used in time series prediction, speech recognition, handwriting recognition and many other applications.

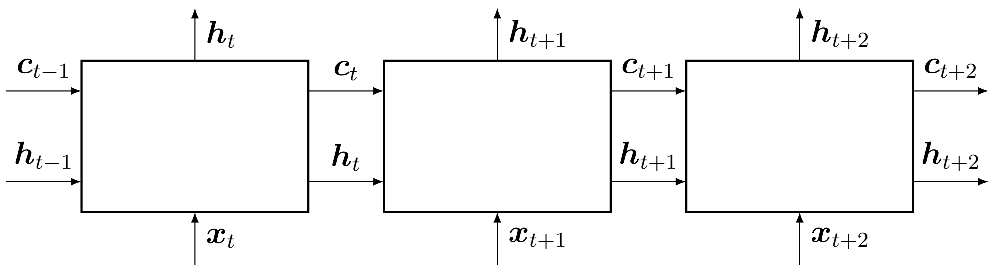


Figure 3.1: Unfolded LSTM network.

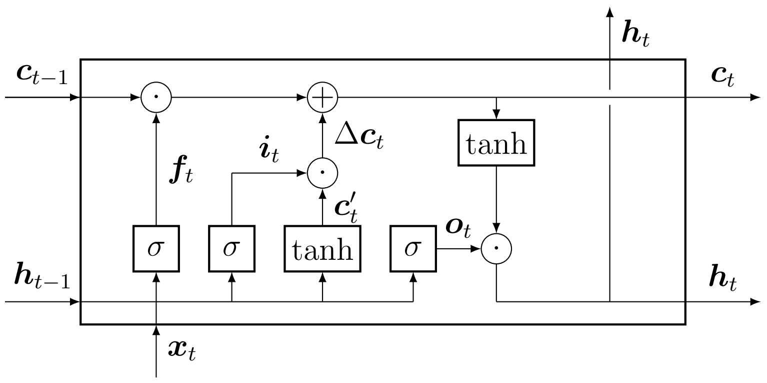


Figure 3.2: LSTM inner structure.

LSTM introduces a more complex inner structure, consisting of the *forget gate*, the *input gate*, the *output gate* and the *cell state*. The cell state at time t is denoted by $\boldsymbol{c}_t$. Input and output at time t are denoted by $\boldsymbol{x}_t$ and $\boldsymbol{h}_t$ respectively. An unfolded LSTM network is depicted in Figure 3.1. Each input can change the cell state, therefore affecting subsequent outputs.

The forget gate determines how much of the previous cell state to consider for the current input. Result $\boldsymbol{f}_t$ of the forget gate, which operates on the concatenation of the current input and previous output, is multiplied by the previous cell state.

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + b_f) \tag{3.1}$$

The input gate determines how the current input changes the cell state. The cell state is then calculated by summing the adjusted previous cell state and the change introduced by the input gate.

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \Delta\boldsymbol{c}_t \tag{3.2}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + b_i) \tag{3.3}$$

$$\boldsymbol{c}'_t = \tanh(\boldsymbol{W}_c \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + b_c) \tag{3.4}$$

$$\Delta\boldsymbol{c}_t = \boldsymbol{i}_t \odot \boldsymbol{c}'_t \tag{3.5}$$

The output gate calculates what the network will output at a specified time t. It combines the cell state with concatenated previous output and the current input.

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + b_o) \tag{3.6}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{3.7}$$

LSTM inner structure is depicted in Figure 3.2. For many years LSTM architecture was the best performing architecture for natural language processing, until the arrival of the Transformer architecture (Vaswani et al., 2017).

3.2 The Transformer

Recurrent models have a fundamental constraint of having sequential inference. This prevents the parallelization of the training process, which is desirable for proper scaling. The Transformer architecture was proposed by (Vaswani et al., 2017) to enable parallelization. The Transformer is a sequence-to-sequence model using multi-head self-attention without recurrent or convolutional layers. The Transformer uses an encoder-decoder structure. Both encoder and decoder stacks are composed of 6 identical layers. This is shown in Figure 3.3, where N is the number of encoder and decoder layers.

Each encoder layer consists of a multi-head self-attention mechanism and a fully connected feed-forward network. There are residual connections which speed up the training around both sublayers followed by layer normalization. Each decoder layer uses an additional sublayer for the encoder output. The self-attention sublayer is modified so that it does not refer to future outputs.

BiLingual evaluation understudy (BLEU) (Papineni et al., 2002) is a way to calculate the quality of a machine-translated text. The Transformer was shown to outperform previous sequence-to-sequence models on machine translation tasks by more than 2 BLEU percentage points while being trained faster than previous models (Vaswani et al., 2017), which was a significant achievement at that time.

3.2.1 Attention

In the basic encoder-decoder architecture, the encoder produces a fixed-length vector which the decoder uses to produce an output sequence. Since fixed-length representation is a bottleneck in this architecture, by allowing the model to focus on parts of the input relevant to predicting the target output, without explicitly constructing them, this bottleneck is mitigated. This is called attention, as first introduced by Bahdanau et al. (2014).

Attention mechanism at a step t uses all encoder outputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ and the current decoder output $\boldsymbol{y}_t$ to calculate the next decoder output $\boldsymbol{y}_{t+1}$. It first calculates attention scores for every $\boldsymbol{x}_i$ with respect to the $\boldsymbol{y}_t$: $score(\boldsymbol{y}_t, \boldsymbol{x}_i)$, $i = \{1, \ldots, n\}$. This

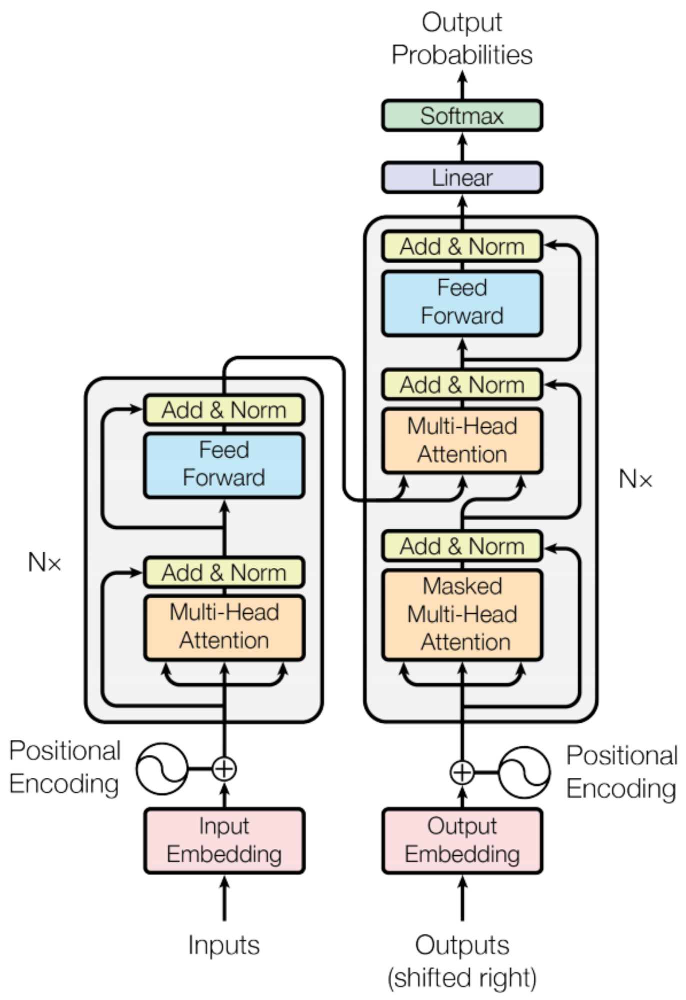


Figure 3.3: The Transformer architecture (Vaswani et al., 2017).

score is used to calculate attention weight w_i for the corresponding $\boldsymbol{x}_i$ using the softmax function across all encoder outputs:

$$w_i = \frac{\exp(score(\boldsymbol{y}_t, \boldsymbol{x}_i))}{\sum_{k=1}^{n} \exp(score(\boldsymbol{y}_t, \boldsymbol{x}_k))} \tag{3.8}$$

The final attention output is a weighted sum of the encoder outputs and the corresponding attention weights:

$$\boldsymbol{y_i} = \sum_{i=1}^{n} w_i \boldsymbol{x}_i \tag{3.9}$$

The *score* function doesn't have to be a specific one. Multiple *score* functions are used in the literature. Bahdanau et al. (2014) used a multi-layer perceptron. Luong et al. (2015) used a bilinear function. The authors of the Transformer (Vaswani et al., 2017) used a simpler method — the scaled dot-product.

The Transformer additionally uses self-attention between each encoder and decoder layer. Self-attention is a mechanism in which each token looks at other tokens to update its weight based on their relevance. The current asking token is called *a query*. Other tokens produce a *key–value* pair each. The key is used to calculate the weight of the corresponding token. The value is used to update the asking token based on the weights. Query, key and value are assigned by computing three linear transformations on n tokens:

$$\boldsymbol{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i \tag{3.10}$$

$$\boldsymbol{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i \tag{3.11}$$

$$\boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i \tag{3.12}$$

where $i = \{1, \dots, n\}$, $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{\frac{d}{h} \times d}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, d is the dimensionality of token embedding and h is the number of attention heads. Dot-product self-attention for output token $\boldsymbol{y}_i$ is then computed as follows:

$$w_i = softmax(\boldsymbol{q}_i^T \boldsymbol{k}_i) \tag{3.13}$$

$$\boldsymbol{y}_i = w_i \boldsymbol{v}_i \tag{3.14}$$

The result is a token representation $\boldsymbol{y}_i \in \mathbb{R}^{\frac{d}{h}}$. The authors of the Transformer additionally scale the dot-product by dividing it with the square root of the embedding dimension, because they suspect it helps push the *softmax* out of extremely small gradients. This division on average compensates for the increase in length produced by added dimensions, as it is known that a unit vector of dimension d has a Euclidean norm $\sqrt{d}$. The scaled dot-product self-attention can be efficiently implemented using matrix multiplication by packing all queries, keys and values to matrices.

Multi-head attention (MHA) concatenates the output of each attention head. The output is then projected through a feed-forward layer back to a vector with d dimensions:

$$\boldsymbol{z} = Concat(\boldsymbol{y}_1, \dots, \boldsymbol{y}_h)^T \boldsymbol{W}_o \tag{3.15}$$

where $\boldsymbol{z} \in \mathbb{R}^d$ is the output of the Transformer block and $\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$ is the weight matrix of the feed-forward layer (Vaswani et al., 2017). Further experiments with multi-head attention were conducted by Voita et al. (2019). They discovered that attention heads played interpretable roles within the model. They were either *positional heads*, *syntactic heads* (shown in Figure 3.4) or *rare token heads*. With a larger number of heads, some heads became less useful and could be removed without noticeably affecting the accuracy of the model. A similar multi-head analysis was conducted by Michel et al. (2019). They have similarly concluded that a large amount of heads is redundant. In some cases, one attention head was enough.

3.2.2 Positional encoding

Since neither recurrence nor convolution is present in the model, position information must be added alongside input of both encoder and decoder stack. The positional

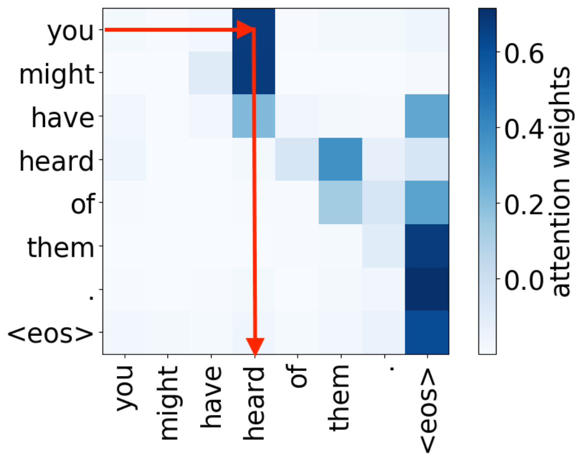


Figure 3.4: A visualization of weights inside a syntactic attention head detecting subject-verb relationship. Taken from Voita et al. (2019).

encodings have the same dimension as the embeddings and are summed with them as shown in Figure 3.3. The authors used sine and cosine in the following way:

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{3.16}$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{3.17}$$

where pos is the position and i is the dimension. Learned positional embeddings were also explored, but yielded comparable results. The authors chose the sinusoidal version "because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training" (Vaswani et al., 2017).

3.3 GPT

Researchers at OpenAI combined unsupervised generative pre-training (GPT) with the Transformer (Radford et al., 2018). Unsupervised pre-training creates a generalized language model, which is then fine-tuned to a specific task using supervised learning. Pre-training is done on a large and diverse corpus to allow the model to collect highly accurate statistical information from language. This allows models to scale as labelling data is a time consuming and expensive process. During unsupervised pre-training on a corpus $\mathcal{U} = \{u_1, \ldots u_n\}$, the objective is to maximize the likelihood of predicting the next token based on k tokens before it:

$$L(\mathcal{U}) = \sum_{i}^{n} \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \boldsymbol{\theta}) \tag{3.18}$$

where $\boldsymbol{\theta}$ is the parameters of the neural network. GPT neural network is made up of 12 blocks of decoder-only transformers with masked self-attention heads. Authors exchanged ReLU for Gaussian Error Linear Unit (GELU) activation function (Hendrycks

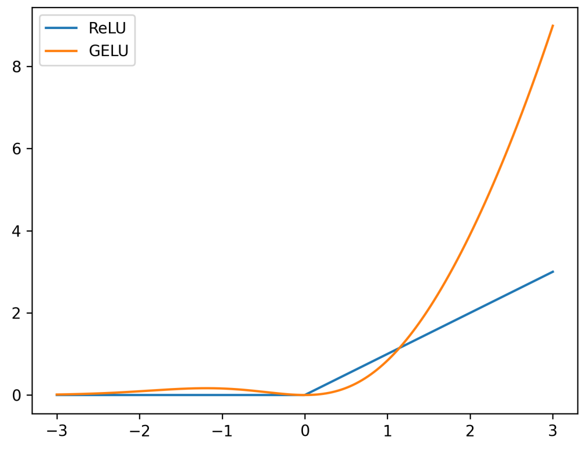


Figure 3.5: Comparison of ReLU and GELU activation functions.

& Gimpel, 2016). GELU is defined as $x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. Their differences are shown in Figure 3.5.

Some of the datasets for natural language understanding tasks that were used to evaluate the model include Story Cloze (Mostafazadeh et al., 2016), Reading Comprehension dataset from Examinations (RACE) (Lai et al., 2017), MultiNLI (Williams et al., 2018) and other GLUE (Wang et al., 2018) datasets. Ablation studies showed that lack of pre-training hinders performance across all studied tasks.

The follow-up research showed that high-capacity models like GPT-2 are capable of state-of-the-art performance on downstream tasks without explicit supervised fine-tuning (so-called zero-shot performance) (Radford et al., 2019). However, GPT-2 zero-shot performance is not satisfactory for real-world problems. By applying further scaling of these models, researchers managed to achieve close to human-level news article generation using GPT-3 (Brown et al., 2020). GPT-3 was later fine-tuned for book summarization using a combination of recursive summarization with human feedback, which yielded sensible summaries of entire books, sometimes matching human-level performance (5% of the books) (Wu et al., 2021).

3.4 BERT

BERT (shorthand for Bidirectional Encoder Representations from Transformers) is a language representation model introduced by (Devlin et al., 2019). The architecture consists of a stack of encoder-only transformer blocks. The input is modified by introducing special tokens [CLS], [SEP] and [PAD]. [CLS] token is used as a beginning of a sequence, [SEP] is used to separate sentences and [PAD] token is used to optionally pad the input to the batch length. The authors describe two BERT models: $\text{BERT}_{\text{BASE}}$

and BERT$_{\text{LARGE}}$. BERT$_{\text{BASE}}$ has 12 transformer blocks and 12 attention heads and is similar in size to GPT. BERT$_{\text{LARGE}}$ has 24 transformer blocks and 16 attention heads. BERT uses GELU activation function.

Training BERT is similar to training GPT. It is done through unsupervised pre-training and supervised fine-tuning. The transformer stack is pre-trained on a large wide-domain corpus. Pre-training takes an input sequence, masks out some words using the [MASK] token (usually 15% of the words) and asks the model to predict the masked out words. This way the model is forced to learn the representations for every word in the sequence in a bidirectional context. The authors refer to this process as *masked language modelling* (MLM) (Devlin et al., 2019). Using a pre-trained model for mask language modelling is very simple. A code snippet is shown in Listing 3.1.

```
from transformers import pipeline

model = pipeline('fill-mask', model='bert-large-uncased')
predictions = model("The incident happened inside the [MASK].")
for prediction in predictions:
    print(prediction['token_str'] + ': ' + str(prediction['score']))
```

Listing 3.1: A simple input with one masked word for BERT$_{\text{LARGE}}$ model.

BERT$_{\text{LARGE}}$ predicts that the masked word in the sentence "*The incident happened inside the [MASK]*" is one of the words shown with their probability scores in Listing 3.2.

```
church: 0.11056565493345 26
building: 0.08972711116075516
house: 0.07968667894601822
school: 0.06394376605749 13
hotel: 0.026174457743763924
```

Listing 3.2: Output of the BERT$_{\text{LARGE}}$ model.

Additionally, pre-training does a second task called *next sentence prediction* (NSP) (Devlin et al., 2019) to improve the model's ability to understand the relationship between sentences. When choosing two sentences for each pre-training input, half of the pairs won't be neighbouring sentences, but sentences chosen at random. The model is then asked to predict whether the sentence pair is neighbouring or random.

Goldberg (2019) analysed the syntactic abilities of the BERT model and determined that it captures the hierarchy-sensitive dependencies and syntactic dependencies very well. Based on previous analysis, Goldberg concluded that BERT did not overly rely on memorising the training data and is instead doing real syntactic generalization.

The pre-trained BERT is good at filling masked words in a sentence. To use this model for other tasks such as token classification, question answering, translation, summarization, text generation and others, the model is fine-tuned on a dataset for this task. After pre-training, a task-specific layer is added to the model and the whole model is then fine-tuned for that specific task (Devlin et al., 2019).

Some authors argue that the $\text{BERT}_{\text{BASE}}$ is heavily overparametrised (Kovaleva et al., 2019). This could explain why some lightweight BERT variants show good results.

3.4.1 BERT-like models

Various modifications of the original BERT model can be found in the literature. What follows is a brief overview of a few of these models.

RoBERTa
Authors of RoBERTa (Liu et al., 2019) found that the original BERT was significantly undertrained. They proposed a modified training process in which training is longer, with increased batch size. They removed the NSP training task and dynamically changed the masking pattern applied to the training data, which resulted in state-of-the-art performance. The introduced modifications showed that MLM-only pre-training can be competitive with other methods. Instead of performing masking once during data preprocessing (static masking) RoBERTa approach generates masking each time sequence is fed to the model. The results showed that dynamic masking is comparable to or slightly better than static masking (Liu et al., 2019).

ALBERT
Two techniques for parameter reduction were proposed by Lan et al. (2020) to decrease the memory needs of BERT and speed up the training. Both BERT and RoBERTa have matching sizes of WordPiece embedding E and hidden layer H. With a large vocabulary size V, the embedding matrix, which has size $V \times E$ ($V \times H$), increases too and causes most of the weights to be updated only rarely during training. The authors reduce the embedding parameters by decomposing a matrix of size $V \times H$ to two matrices of size $V \times E$ and $E \times H$. This reduction becomes meaningful when $H \gg E$. They call this technique *factorized embedding parameterization.* The second parameter reduction technique is to share all parameters across layers. The authors compared the results with Deep Equilibrium Models, which have similar parameter sharing, but the results differed as input and output embedding of a certain layer in ALBERT didn't converge (Bai et al., 2019; Lan et al., 2020). The authors also introduce an alternative for the NSP task called sentence-order prediction (SOP) in which the model needs to predict if two given sentences are in the correct order.

BART
The authors of BART (Lewis et al., 2019) introduced a BERT modified to be a de-

noising autoencoder which reconstructs corrupted text. In addition to token masking, BART is pre-trained on four new pre-training tasks: token deletion, text infilling, sentence permutation and document rotation. In token deletion, some tokens are randomly deleted and the model needs to correctly predict which tokens are missing and where to fill them in. Text infilling entails masking of a span (multiple tokens), therefore teaching the model to fill in more tokens when necessary. Sentences in a document are randomly permuted in sentence permutation, forcing the model to learn to detect this. Finally, document rotation entails finding the original beginning in a rotated document, i.e. document beginning with a chosen random token from the document. The authors conclude that token masking and token deletion are the most important pre-training tasks. BART was shown to perform similar to RoBERTa on discriminative tasks, but was better at a number of text generation tasks (Lewis et al., 2019).

3.5 SlovakBERT

SlovakBERT is one of the first two Slovak-only models trained on a large corpus that appeared in September 2021 (the other being FERNET-cc_sk) (Lehečka & Švec, 2021; Pikuliak et al., 2021). The model has RoBERTa architecture and was trained on an undisclosed Web-crawled corpus. The corpus consists of Wikipedia text, Open Subtitles and OSCAR corpus (5.3 GB in total from public datasets). Additionally, crawled Slovak webpages stripped of HTML tags were added to the corpus. Post-processing and deduplication yielded 19.35 GB of text. SlovakBERT uses a BPE tokenizer (for BPE algorithm summary see Appendix A) and has a vocabulary consisting of 50264 tokens. The authors evaluated SlovakBERT on 4 downstream tasks, namely part-of-speech tagging, semantic textual similarity, sentiment analysis and document classification. In all 4 downstream tasks SlovakBERT confirms or exceeds state-of-the-art results.

Part-of-speech (POS) tagging is a task to mark up words in a text as corresponding to their particular part of speech. The dataset used for fine-tuning POS tagging was the Slovak Dependency Treebank from the Universal Dependencies dataset (Nivre et al., 2020). Probing the fine-tuned model showed that the morphosyntactic information needed for POS tagging was located mainly in the middle part of the model.

Semantic textual similarity (STS) is a task in which semantic similarity between pairs of sentences is measured. As no native Slovak STS dataset existed, the authors translated English STS datasets, namely STSBenchmark (Cer et al., 2017), SICK (Marelli et al., 2014), SNLI (Bowman et al., 2015) and MNLI (Williams et al., 2018). They decided to use sentence embeddings with cosine similarity to assign a specific semantic similarity value. Probing of the fine-tuned model showed that the last layers were the best performing, unlike the POS fine-tuned model.

Sentiment analysis is a task where a positive, neutral or negative label is assigned to a given text based on its sentiment. The authors post-processed a Twitter dataset (Mozetič et al., 2016) and used it to fine-tune the model. SlovakBERT was shown to beat previous models on 3-class sentiment analysis on this dataset.

Document classification is a task where a document is assigned to one of the 6 news categories. To fine-tune the model, the authors used the Slovak Categorized News corpus by (Hladek et al., 2014). SlovakBERT narrowly beats all previous models, achieving the highest F1 score.

3.6 Knowledge distillation

It has been shown that large-scale deep models achieve great performance, however, it is challenging to deploy these models on mobile and embedded devices due to computational complexity and storage requirements. Multiple compression and acceleration techniques are used to alleviate these problems, one of them being *knowledge distillation* (KD) (Gou et al., 2021; Hinton et al., 2015). In knowledge distillation, a smaller student model is supervised by a larger teacher model. The student model is forced to mimic the teacher model in order to achieve similar results while having fewer parameters.

There are different types of knowledge that can be passed from the teacher model to the student model. The most simple knowledge is the final output of the teacher model. This type of knowledge is called *response-based knowledge* (Gou et al., 2021). BERT uses a softmax output layer to convert the logit z_i into a probability for that token $P(z_i)$ by comparing z_i with other logits:

$$P(z_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3.19}$$

Usually, the parameter T (called *temperature*) is set to 2. Higher values make the token probability distribution softer (Hinton et al., 2015). The student model loss function is usually a linear combination of the distillation loss L_D and the standard student loss L. More specifically, $L_D = \mathcal{L}_{\mathrm{KL}}(P(\boldsymbol{z}_t), P(\boldsymbol{z}_s))$, where $\mathcal{L}_{\mathrm{KL}}$ is Kullback-Leiber (KL) divergence, $\boldsymbol{z}_t$ are teacher logits and $\boldsymbol{z}_s$ are student logits. The standard student loss employs cross-entropy between the ground truth and the output probabilities of the student model: $L = L_{\mathrm{CE}}(\boldsymbol{y}, P(\boldsymbol{z}_s))$, where $\boldsymbol{y}$ is the ground truth.

Another approach is to provide the student with outputs of the intermediate layers of the teacher model in order to help it learn similar inner representations. This type of knowledge is called *feature-based knowledge.* In this case, the student loss function is different from response-based distillation. Instead of distillation loss being KL divergence, it is a similarity function used to match the layer activations, also

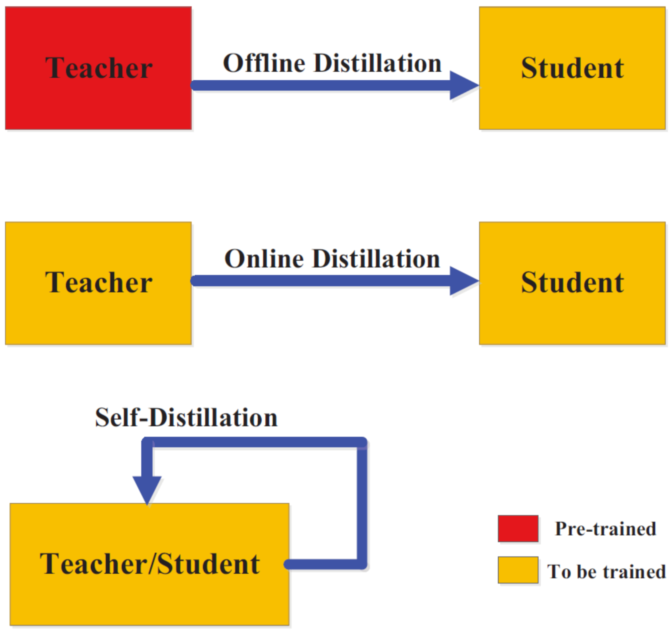


Figure 3.6: Three main categories of distillation schemes (Gou et al., 2021).

called feature maps, of teacher and student models. When feature maps are not of the same shape (the case when student layers are thinner) additional transformations are applied to bring them to a common representation (Gou et al., 2021).

The knowledge that captures the relationship between feature maps can also be used to train a student model. This approach is called *relation-based knowledge*. It is usually modelled as a correlation between feature maps.

3.6.1 Distillation schemes

The distillation schemes can be divided into three categories based on the fact whether the teacher model is trained simultaneously with the student model or not: *online distillation*, *offline distillation* and *self-distillation*.

In offline distillation, a pre-trained teacher model is distilled into a student model. This scheme is useful when we already have a large teacher model. In online distillation, both the teacher and the student model are updated simultaneously, thus allowing the scheme to be end-to-end trainable. This scheme is useful when a large high-performance teacher model is not available. The self-distillation uses the same network for the teacher and student models. This scheme allows us to distil features from the deeper layers to the shallow layers of the network. The schemes are shown in Figure 3.6 (Gou et al., 2021).

3.6.2 Non-standard distillation algorithms

Acquiring knowledge from the teacher model can be done in various ways. It is also possible for one student to have multiple teachers.

Adversarial distillation was proposed as a way to mitigate the problem of non-availability of data used to train the teacher model. Pseudo-examples are generated adversarially and those are used to match the student to the teacher (Micaelli & Storkey, 2019). Generative adversarial networks (GANs) can also be used to augment the training dataset or to generate hard examples for knowledge transfer (Gou et al., 2021).

Multi-teacher distillation employs multiple teachers. Each teacher could potentially have been trained on a different corpus, thus the student can benefit from the diverse knowledge that each of the teachers brings. Besides averaging teacher logits and hidden states, it can be challenging to find appropriate functions to merge teacher outputs. Alternatively, each teacher can provide different types of knowledge, for example, one can provide logits while the other provides feature knowledge in the form of hidden states (Chen et al., 2019). Initializing the student network with layers from multiple teachers could also have a potential benefit for the student.

Quantized distillation reduces the computation complexity of neural networks by converting high-precision networks using 32-bit floating-point weights into low-precision networks using only 8-bit or even only 2-bit floating-point weights. Knowledge distillation enables these low-precision models to achieve comparable performance to high-precision teacher models (Gou et al., 2021). Distillation loss can also be calculated by first converting the teacher feature maps to lower precision and then comparing them to student feature maps (Mishra & Marr, 2017).

3.6.3 BERT with knowledge distillation

Various approaches have been proposed on how to use knowledge distillation algorithms on BERT models. What follows is a brief overview of a few of them.

DistilBERT

The authors of DistilBERT (Sanh et al., 2020) applied knowledge distillation during the pre-training to obtain a distilled version of the BERT model which is smaller by 40% and faster by 60% while maintaining 97% of model accuracy. The training objective was to minimize the triplet loss function consisting of distillation loss, masked language modelling loss and cosine embedding loss. Student architecture is similar to the teacher architecture, albeit token-type embedding and pooler layers are removed, the number of layers is cut in half and the student is initialized with every second layer from the teacher model. The authors used dynamic masking and the next sentence prediction was removed from training. Investigation into the influence of various components of

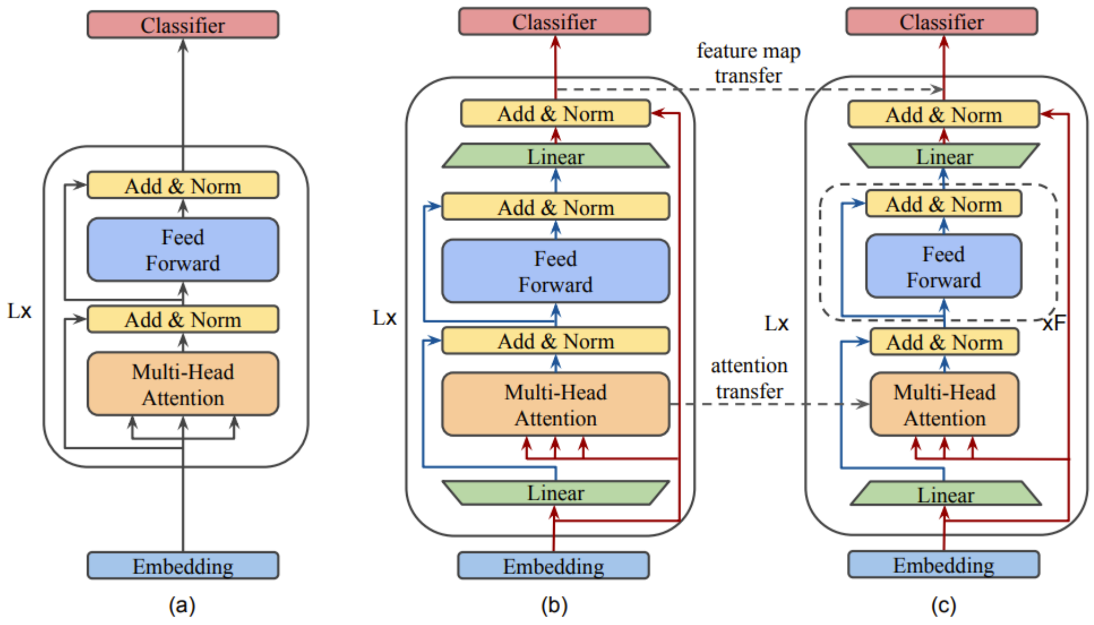


Figure 3.7: a) BERT b) Inverted bottleneck BERT, c) MobileBERT (Sun et al., 2020)

the triplet loss showed that masked language modelling loss had little impact, while two other components were dominant.

TinyBERT

A two-stage learning framework (Jiao et al., 2019) combines general distillation with task-specific distillation. Task-specific distillation enables the unnecessary knowledge for these tasks to be pruned from the distilled general model, thus reducing model size significantly. TinyBERT with four layers can retain 96.8% of the original performance on a concrete downstream task while having only 14.5M parameters — 13.3% of the original 109M parameters in BERT$_{\mathrm{BASE}}$. Multiple ablation studies conducted by the authors showed that general distillation contributes more to some specific datasets that require linguistic acceptability judgements, such as the Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2018).

MobileBERT

Bottlenecks inside BERT encoder blocks were introduced by Sun et al. (2020) to reduce the dimensionality inside encoder blocks. This was realised by adding two linear layers to convert 512 dimensions to 128 dimensions and later back to 512 dimensions. Additionally, to balance the parameters in attention layers and linear layers, the authors added more linear layers inside each block as shown in Figure 3.7.

To alleviate problems with training a narrow network, the authors first trained a BERT with inverted bottleneck layers to convergence and then performed feature map and attention transfer to the model with MobileBERT architecture. The MobileBERT was shown to outperform all previous compressed and distilled models with smaller or comparable sizes (Sun et al., 2020).

3.7 Evaluation of language models

It can be cumbersome to directly compare pre-trained general language models. They are usually compared by fine-tuning them on several downstream tasks and comparing their scores on those tasks. What follows is a brief summarization of two widespread evaluation benchmarks for language models.

3.7.1 GLUE

The General Language Understanding Evaluation benchmark (GLUE) is a collection of nine sentence-understanding tasks (Wang et al., 2018). The Corpus of Linguistic Acceptability (**CoLA**) and The Stanford Sentiment Treebank (**SST-2**) are both single sentence tasks, in which the language model should correctly predict the grammatical acceptability of a sentence or sentiment of a given sentence, respectively. GLUE also contains three tasks in which the model needs to correctly predict whether the two sentences are semantically equivalent. The Microsoft Research Paraphrase Corpus (**MRPC**) contains sentences from news sources, the Quora Question Pairs (**QQP**) contains question pairs from question-answering website *Quora*, while the Semantic Textual Similarity Benchmark (**STS-B**) contains sentences from news headlines, image captions and other sources. Additionally, GLUE contains four inference tasks. The Multi-Genre Natural Language Inference Corpus (**MNLI**) and The Recognizing Textual Entailment (**RTE**) are tasks in which the model needs to correctly predict whether a given premise and hypothesis sentences are entailed, contradicted or neither. Question-answering NLI (**QNLI**) is a task made by converting the Stanford Question Answering Dataset (**SQuAD**), which is not part of GLUE, to context-question pairs, where the model is tasked to correctly predict whether the context contains the answer to the given question. The Winograd Schema Challenge (**WNLI**) is a task in which the model gets a sentence with a pronoun and must choose the referent of that pronoun from given choices.

3.7.2 SuperGLUE

The SuperGLUE benchmark contains eight language understanding tasks (Wang et al., 2019). Boolean questions (**BoolQ**) is a question answering dataset containing paragraphs and accompanying yes/no questions, which the model needs to correctly answer either yes or no. The Choice of Plausible Alternatives (**CoPA**) is a reasoning task in which the model chooses between two alternatives that answer the given question about a given premise sentence. Multi-Sentence Reading Comprehension (**MultiRC**) is a task in which the model needs to determine the truthfulness of the answers to the question that is related to the given paragraph. Reading Comprehension with Commonsense Reasoning Dataset (**ReCoRD**) is a task in which the model needs to correctly fill in the masked word in a question related to the given paragraph. Recognizing Textual Entailment (**RTE**) is a task in which a text-hypothesis pair is to be classified either as *entailment* or *not entailment*. Word-in-Context (**WiC**) is a task in which the model needs to predict if the polysemous word appearing in two sentences is having the same sense or not. In Winograd Schema Challenge (**WSC**) the model needs to correctly predict the referent of a pronoun in the given sentences while having multiple choices present. Finally, CommitmentBank (**CB**) is a classification task in which the model needs to correctly determine if a person who wrote the text containing an embedded clause is committed to the truth of the clause.

Chapter 4

Proposed solution

We propose six different experimental setups for distilling SlovakBERT. Two experiments contain training of a 6-layer student model, while four contain training of a 4-layer student model. One 4-layer student model is trained using two teacher models. All students are based on the RoBERTa architecture with the same hidden and intermediate size as SlovakBERT. We evaluate all experiments on four downstream tasks: NER, UPOS, STS and BoolQ. Datasets for STS and BoolQ are machine-translated into the Slovak language.

4.1 Dataset

Effective pre-training of a language model requires a very large text corpus. The size of the clean text from Slovak Wikipedia is currently around a couple of hundred megabytes. SlovakBERT was trained on almost 20 gigabytes of clean text. An effective knowledge distillation requires at least a couple of gigabytes of clean text. This requirement forced us to find alternative larger sources of Slovak text. In addition, the Wikipedia style of writing is not general enough for wide-domain applications and models trained or distilled only on Wikipedia texts may underperform on some downstream tasks.

Every downstream task requires finding an appropriate training dataset. Slovak datasets for downstream tasks are not very common. We resort to machine-translating two of the mentioned downstream datasets.

4.1.1 Common Crawl

Common Crawl is a non-profit organization that maintains a copy of the internet data, called the Common Crawl corpus. It provides it to researchers, companies and others for research purposes (Common Crawl, 2022). The Common Crawl corpus has been accumulating data since 2008 by resorting to *web crawlers*, which are programs that systematically browse the web and download the pages they encounter. Crawlers are nowhere near to visiting every available web page. Nonetheless, they can accumulate a significant amount of data this way. The current size of the uncompressed data is more than 360 TB. It contains raw web page data, extracted metadata and extracted text.

4.1.2 C4 dataset

C4 dataset is a large collection of web crawled text that was put together by Google researchers in 2020 (Raffel et al., 2020; Xue et al., 2021). The authors did not offer it for download, but instead, they published open-source tools, which can re-create it from the original Common Crawl data. The C4 dataset was re-created by people from a non-profit research institute AI2, created by Microsoft co-founder Paul Allen (AI2, 2022; GitHub, 2021). This dataset contains 5 sets of data:

en	305 GB
en.noclean	2.3 TB
en.noblocklist	380 GB
realnewslike	15 GB
multilingual	9.7 TB

Multilingual set is divided into specific languages. We are interested in Slovak data inside the multilingual set. This data comes in compressed JSON format in which each JSON object contains: *text*, *timestamp* and *url* (of the website from which it originated). After decompressing and removing the *timestamp* and *url* we are left with more than 53 GB of clean Slovak text data. We use a 1,9 GB subset for knowledge distillation[1].

Besides this dataset, we needed datasets that would be used to fine-tune and evaluate SlovakBERT and distilled student models on downstream tasks. These fine-tuning datasets are significantly smaller in size.

4.1.3 WikiANN dataset

WikiANN (Pan et al., 2017; Rahimi et al., 2019) is a dataset for supervised named-entity recognition (NER) training. NER is a task that aims to locate and classify named entities from unstructured text into pre-defined categories. WikiANN dataset contains labels for persons, locations and organizations, therefore supporting three categories. Optionally, unclassified entities can be considered as belonging to the background class. Slovak subset contains 20,000 train, 10,000 dev and 10,000 test examples.

[1]In particular, the subset consists of all files that start with *c4-sk.tfrecord-005*.

We used the macro-averaged F_1 score to evaluate the performance of models on this downstream task. It is most commonly used to assess the performance of models on tasks with multiple binary labels or multiple classes. Values range from 0 (worst) to 1 (best). Macro-averaged F_1 score is calculated by averaging all single-class F_1 scores. Single-class F_β scores are calculated as follows:

$$F_\beta = \frac{(1+\beta^2)\cdot TP}{(1+\beta^2)\cdot TP + \beta^2 \cdot FN + FP} \tag{4.1}$$

where TP is the number of true positive examples, FN is the number of false negative examples (type II error) and FP is the number of false positive examples (type I error). Single-class F_1 score is therefore:

$$F_1 = \frac{2\cdot TP}{2\cdot TP + FN + FP} \tag{4.2}$$

A visualization of example output from a model fine-tuned on WikiANN dataset is shown in Figure 4.1.

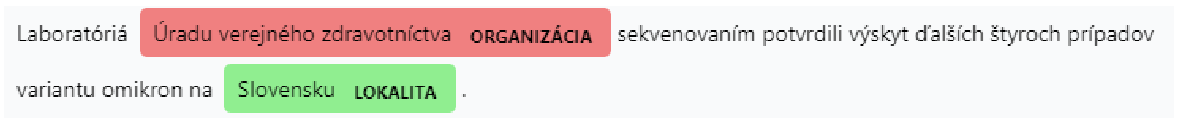

Laboratóriá
Úradu verejného zdravotníctva ORGANIZÁCIA
sekvenovaním potvrdili výskyt ďalších štyroch prípadov
variantu omikron na
Slovensku LOKALITA
.

Figure 4.1: A visualization of a correct NER labelling of an organization (red) and a location (green).

4.1.4 Universal dependencies dataset

Universal dependencies is a collection of treebanks for various world languages (Nivre et al., 2020). A treebank is a collection of texts that contain annotations for syntactic or semantic sentence structure. Universal dependencies contain universal part-of-speech tags for the Slovak language (Zeman, 2017). A list of all tags can be found in the Appendix A. We used the macro-averaged F_1 score to evaluate the performance of models on this downstream task.

A visualization of example output from a model fine-tuned on this dataset is shown in Figure 4.2.

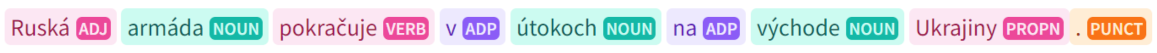


Figure 4.2: A visualization of UPOS tagging in a Slovak sentence.

4.1.5 Translated STSB dataset

STSBenchmark is a dataset used to fine-tune language models on semantic textual similarity (STS) task in which a model needs to predict how similar is the semantic meaning of two given sentences (Cer et al., 2017). The output is a floating-point number between 0 and 5, where 5 means the highest similarity. A more detailed guide for numerical labels is shown in Table 4.1.

Score	Explanation
5	The two sentences are completely equivalent, as they mean the same thing.
4	The two sentences are mostly equivalent, but some unimportant details differ.
3	The two sentences are roughly equivalent, but some important information differs/missing.
2	The two sentences are not equivalent but share some details.
1	The two sentences are not equivalent but are on the same topic.
0	The two sentences are completely dissimilar.

Table 4.1: A guide for sentence similarity scores.

The dataset contains 5,749 train, 1,500 dev and 1,379 test examples. We translated the dataset to Slovak using the English-Slovak translation model *opus-mt-en-sk* by Helsinki-NLP (2020). The pre-trained models are usually fine-tuned to generate sentence embeddings that are then compared using scaled cosine similarity. Fine-tuned models can be evaluated by looking either at the Pearson's or Spearman's rank correlation coefficient between their outputs and the ground truth. Former measures the linear correlation between two variables, while the latter measures how well the relationship between two variables can be described using a monotonic function. Intuitively, we want the scores to be increasing with sentences that are more similar, but a good enough model does not necessarily have to return the same scores as another good model. The Spearman's rank correlation coefficient fits this intuition.

4.1.6 Translated BoolQ dataset

In the Boolean questions (BoolQ) (Clark et al., 2019) dataset each entry contains a text passage, a question related to that passage and a yes/no answer to this question. The models are fine-tuned using this dataset to learn to extract simple yes/no answers from the given passages. The dataset contains 15,942 examples. We translated this dataset to Slovak using *opus-mt-en-sk* model. We show an example of a translated question-passage pair:

passage: Pitie na verejnosti v Dánsku je vo všeobecnosti legálne. Zákon zakazuje narušiť "verejné právo a poriadok". Preto je všeobecne prijímaná spotreba. Niekoľko kaviarní majú vonkajšie služby v rovnakých zónach.
question: Môžete piť alkohol na verejnosti v Denmark?

Some limitations of machine translation can be seen in the example. The names of the countries are sometimes not translated correctly, some sentences lose information and some verbs may contain wrong suffixes. Nonetheless, we were satisfied with the translation quality and used it to fine-tune models on the BoolQ task. BoolQ authors note that the best performance is achieved when fine-tuning the model previously fine-tuned on MultiNLI (Clark et al., 2019).

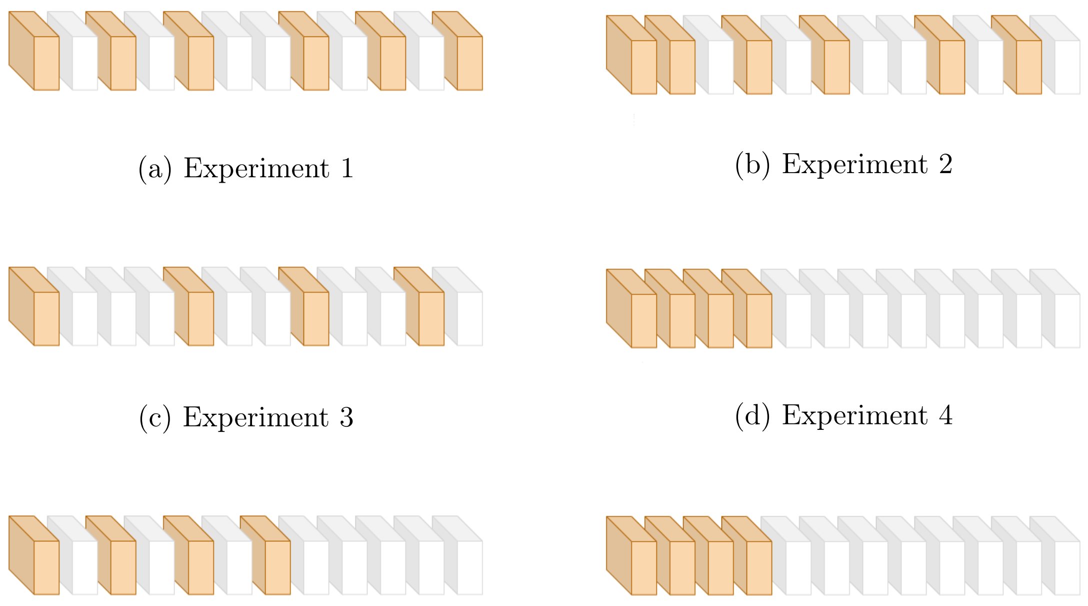
4.2 Experiments

We performed experiments with different weights of loss function components and different starting student weights initialization. The weights were chosen in a way to test whether the principal knowledge is located across the whole network or just in specific parts, such as the beginning. An overview of the experiment configurations is shown in Table 4.2. Student weight initialization is shown in Figure 4.3.

	KL divergence	Cross-entropy	Cosine embedding	Weight init
Experiment 1	0.625	0.25	0.125	[1, 3, 5, 8, 10, 12]
Experiment 2	0.625	0.25	0.125	[1, 2, 4, 6, 9, 11]
Experiment 3	0.6	0.2	0.2	[1, 5, 8, 11]
Experiment 4	0.7	0.2	0.1	[1, 2, 3, 4]
Experiment 5	0.7	0.2	0.1	[1, 3, 5, 7]
Experiment 6	0.7	0.2	0.1	[1, 2, 3, 4]

Table 4.2: Experiment configurations. The *KL divergence*, *Cross-entropy* and *Cosine embedding* represent the weight of each of these components in the final loss function that the distilled model is trained to optimize. The *Weight init* column represents the weight layers that are initialized from the layers of SlovakBERT as illustrated in Figure 4.3.

All experiments used the AdamW optimizer (Loshchilov & Hutter, 2017) with an epsilon value of $1 \cdot 10^{-6}$. This optimizer is very similar to Adam but has decoupled weight decay. The starting learning rate was $5 \cdot 10^{-4}$, while the maximum gradient norm was set to 5.0. We used 50 steps for gradient accumulation, while the batch size was set



(a) Experiment 1

(b) Experiment 2

(c) Experiment 3

(d) Experiment 4

(e) Experiment 5

(f) Experiment 6

Figure 4.3: A visualization of the student weights initialization. Each block represents one Transformer encoder. The leftmost blocks are the input blocks. Each experiment has its configuration outlined in Table 4.2.

to 12. Sentences of similar lengths were grouped together in batches. Sequences that were longer than the maximum position embedding size were split (27,915 in total). Sequences shorter than 11 tokens were removed from the training data (1,051,944 in total). Finally, 6,861,437 sequences were used for training. The proportion of masked tokens was set to the most common value of 15%. We trained for 3 epochs on a dataset of 1.9 GB in size. We applied smoothing to masked language modelling to emphasize rare tokens. The temperature value in knowledge distillation was set to 2 in all experiments.

Student models all had RoBERTa architecture with frozen positional and token type embeddings. Student models had hidden size (768) and intermediate size (3,072) the same as in SlovakBERT. Dropout in hidden and attention layers was set to 0.1.

Experiment 6 had two teacher models, namely models trained in Experiment 1 and Experiment 2. To be able to calculate the student loss, the outputs of the teachers need to be merged somehow. We simply calculated average values for logits and hidden states respectively. This enables any number of teachers to be used in future experiments.

The first five experiments took around 30 hours each, while the last experiment with two teachers took around 53 hours. Knowledge distillation was done on one Nvidia 3090 GPU with 24 GB of VRAM. All knowledge distillation experiments were conducted with the help of the `transformers` library (Wolf et al., 2020).

4.2.1 Fine-tuning

We utilized `transformers` library for fine-tuning models on **NER** and **UPOS** tasks. Fine-tuning ran for 10 epochs for NER and 5 epochs for UPOS, with linearly scheduled learning rate starting at $5 \cdot 10^{-5}$. We used AdamW optimizer with default epsilon ($1 \cdot 10^{-8}$) and beta values (0.9, 0.999). We used batch size 32 for training. For **STS** task we used `sentence-transformers` library (Reimers & Gurevych, 2019) for fine-tuning. The training ran for 4 epochs with batch size 32. Models were fine-tuned on **BoolQ** for 10 epochs with batch size 16 and learning rate $1 \cdot 10^{-5}$. We utilized gradient accumulation and updated the weights after every 10 batches. Both STS and BoolQ fine-tuning also used AdamW optimizer.

Chapter 5

Results

Results of the conducted experiments are shown in Table 5.1. We also compare the FERNET-cc_sk model with the SlovakBERT. The two models show differences in performance on four downstream tasks. SlovakBERT achieves better performance on part-of-speech tagging and boolean questions, while the FERNET-cc_sk performs better on named-entity recognition and semantic textual similarity. Nevertheless, the differences are mostly small.

Model	**NER** (Macro-F1)	**POS** (Macro-F1)	**STS** (Spearman)	**BoolQ** (Accuracy)	# **Params**
SlovakBERT	0.939	**0.983**	0.781	**0.709**	124M
FERNET-cc_sk	**0.941**	0.980	**0.788**	0.663	162M
Experiment 1	0.929	0.976	0.713	0.649	82M
Experiment 2	**0.931**	**0.979**	0.734	0.662	82M
Experiment 3	0.907	0.972	0.720	0.646	67M
Experiment 4	0.916	0.974	**0.743**	**0.668**	67M
Experiment 5	0.915	0.973	0.740	0.645	67M
Experiment 6	0.916	0.975	0.693	0.642	67M

Table 5.1: Distillation results of various tested configurations on four downstream tasks. Each entry is the mean of at least 3 runs. The used score for each task is shown in brackets. Higher is better.

Despite learning from two different teachers, the student model from Experiment 6 slightly underperforms student models from other experiments. This is most probably caused by the high nonlinearity of the parameter space combined with the averaging teacher logits and hidden states. Additional experiments with SlovakBERT and FERNET-cc_sk as two teacher models did not yield results, because of the great dif-

Model	300 examples (s)	Single example (ms)
fernet-cc-sk	3.88	12.93
slovakbert	3.58	11.93
Experiment 1	2.35	7.83
Experiment 2	2.33	7.77
Experiment 3	1.85	6.17
Experiment 4	1.86	6.21
Experiment 5	1.86	6.20
Experiment 6	1.90	6.33

Table 5.2: Inference speed tested using Pipeline from the `transformers` library (Wolf et al., 2020).

Model	Size in MB
slovakbert	476
fernet-cc-sk	624
Experiment 1, 2	313
Experiment 3, 4, 5, 6	260

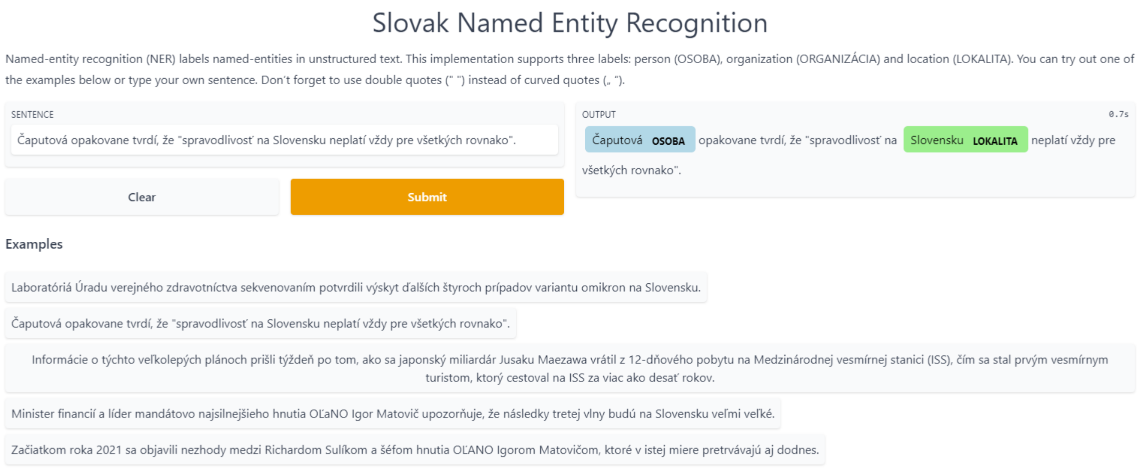
Table 5.3: The size of the models saved in PyTorch format (Paszke et al., 2019).

ficulty in combining mixed-vocabulary teachers. We did not find an elegant way to map a vocabulary the size of 100,000 from FERNET-cc_sk to a vocabulary the size of 50,264 from SlovakBERT.

Another interesting phenomenon can be observed in the performance of the student from Experiment 4, which obtained the best results on both the semantic text similarity as well as the boolean questions tasks, the latter being slightly better than the second teacher model FERNET-cc_sk. We hypothesize that this might be thanks to the model being initialized from the first four layers of the teacher model. This would suggest that it is indeed these layers that hold the information necessary for reasoning over longer sequences, which both of these tasks require.

In summary, when we look at the scores of the 6 layer model, we get from 91% to 99% of the original performance while having around 35% fewer parameters. Distilled models with 4 layers retained similar performance while having 46% fewer parameters. In absolute numbers that is 82 million parameters for 6-layer models and 67 million parameters for 4-layer models, while the original SlovakBERT has 124 million parameters. This shows that distillation is a viable approach to decreasing model size with minimal impact for non-English languages even with a low amount of training data.

In addition to testing performance on four downstream tasks, we also compare inference speed and memory requirements. Inference time is tested by both measuring



Slovak Named Entity Recognition

Named-entity recognition (NER) labels named-entities in unstructured text. This implementation supports three labels: person (OSOBA), organization (ORGANIZÁCIA) and location (LOKALITA). You can try out one of the examples below or type your own sentence. Don't forget to use double quotes (" ") instead of curved quotes („ ").

SENTENCE

Čaputová opakovane tvrdí, že "spravodlivosť na Slovensku neplatí vždy pre všetkých rovnako".

OUTPUT 0.7s

Čaputová OSOBA opakovane tvrdí, že "spravodlivosť na Slovensku LOKALITA neplatí vždy pre všetkých rovnako".

Examples

Laboratóriá Úradu verejného zdravotníctva sekvenovaním potvrdili výskyt ďalších štyroch prípadov variantu omikron na Slovensku.

Čaputová opakovane tvrdí, že "spravodlivosť na Slovensku neplatí vždy pre všetkých rovnako".

Informácie o týchto veľkolepých plánoch prišli týždeň po tom, ako sa japonský miliardár Jusaku Maezawa vrátil z 12-dňového pobytu na Medzinárodnej vesmírnej stanici (ISS), čím sa stal prvým vesmírnym turistom, ktorý cestoval na ISS za viac ako desať rokov.

Minister financií a líder mandátovo najsilnejšieho hnutia OĽaNO Igor Matovič upozorňuje, že následky tretej vlny budú na Slovensku veľmi veľké.

Začiatkom roka 2021 sa objavili nezhody medzi Richardom Sulíkom a šéfom hnutia OĽANO Igorom Matovičom, ktoré v istej miere pretrvávajú aj dodnes.

Figure 5.1: The deployed SlovakBERT NER model is available for inline inference.

the time needed to process one example and 300 examples. Decreases in inference time and size in memory are very promising. Results are shown in Table 5.2 and Table 5.3, respectively.

We publish the machine-translated datasets on HuggingFace. We also publish the distilled models for further study at https://huggingface.co/crabz. Fine-tuned Slovak-BERT NER model is published online and can be accessed and used for inline inference at https://huggingface.co/spaces/crabz/sk-ner. A screenshot from the UI interface of the deployed model is shown in Figure 5.1.

Appendix B contains an overview of the electronic attachment that is provided with this work.

Chapter 6

Conclusion

We showed that language-specific knowledge distillation is a viable technique for lowering the model size while retaining nearly all of its original performance. Furthermore, we demonstrated the importance of choosing the right initial student weights. We achieved from 91% to 99% of the original performance across all four downstream tasks, while the distilled models at the same time had up to 46% fewer parameters. We further demonstrated that the principal knowledge needed for a specific task can be located in specific layers of the original model, therefore enabling us to initialize the student with the weights from these layers and consequently achieve better task-specific performance. In addition, we showed that averaging logits and hidden states while performing knowledge distillation from multiple teachers, which had seen the same training dataset, did not provide an advantage to the student model.

Machine translation has some limitations and those can be seen throughout the translated datasets. This negatively impacts model performance. Translation models with a larger maximum input size could provide better translations for longer sentences. Regardless, an expert human translation of these datasets is very desired.

A promising further work would be to apply techniques introduced in TinyBERT and MobileBERT to the distillation setup for SlovakBERT. This could potentially yield models with even fewer parameters with comparable performance on downstream tasks. Another research direction would be to implement mixed-vocabulary distillation with SlovakBERT and FERNET-cc_sk as teachers. These two models had seen slightly different datasets and could provide useful information for the student which could potentially be even better than its teachers on some downstream tasks.

We published distilled models, translated datasets and training scripts on HuggingFace and in the electronic attachment for further study.

References

AI2. (2022). Ai2 [Accessed: May 6, 2022]. https://allenai.org/

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*.

Bai, S., Kolter, J. Z., & Koltun, V. (2019). Deep Equilibrium Models. https://doi.org/10.48550/arxiv.1909.01377

Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015). A Large Annotated Corpus for Learning Natural Language Inference. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., & Specia, L. (2017). Semeval-2017 task 1: Semantic Textual Similarity — Multilingual and Cross-Lingual Focused Evaluation. *arXiv preprint arXiv:1708.00055*.

Chen, X., Su, J., & Zhang, J. (2019). A Two-Teacher Framework for Knowledge Distillation. *International Symposium on Neural Networks*, 58–66.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., & Toutanova, K. (2019). BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. *arXiv preprint arXiv:1905.10044*.

Common Crawl, o. (2022). Common Crawl [Accessed: May 6, 2022]. https://commoncrawl.org/

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, *14*(2), 179–211.

Gage, P. (1994). A New Algorithm for Data Compression. *The C Users Journal Archive*, *12*, 23–38.

GitHub. (2021). C4 GitHub Discussion [Accessed: May 6, 2022]. https://github.com/allenai/allennlp/discussions/5056

Goldberg, Y. (2019). Assessing BERT's Syntactic Abilities. *arXiv preprint arXiv:1901.05287*.

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge Distillation: A Survey. *International Journal of Computer Vision*, *129*(6), 1789–1819.

Helsinki-NLP. (2020). Helsinki-NLP/opus-mt-en-sk · Hugging Face. https://huggingface.co/Helsinki-NLP/opus-mt-en-sk

Hendrycks, D., & Gimpel, K. (2016). Gaussian Error Linear Units (GELUs). https://doi.org/10.48550/arxiv.1606.08415

Hinton, G., Vinyals, O., Dean, J., et al. (2015). Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, *2*(7).

Hladek, D., Stas, J., & Juhar, J. (2014). The Slovak Categorized News Corpus. *LREC*, 1705–1708.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

Honnibal, M., & Montani, I. (2017). *SpaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing* [To appear].

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, *2*(5), 359–366.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., & Liu, Q. (2019). TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv preprint arXiv:1909.10351*.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Korzhenkov, A. (2010). *Zamenhof: The Life, Works and Ideas of the Author of Esperanto*. Mondial.

Kovaleva, O., Romanov, A., Rogers, A., & Rumshisky, A. (2019). Revealing the Dark Secrets of BERT. *arXiv preprint arXiv:1908.08593*.

Lai, G., Xie, Q., Liu, H., Yang, Y., & Hovy, E. (2017). RACE: Large-Scale ReAding Comprehension Dataset From Examinations. *arXiv preprint arXiv:1704.04683*.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations. *arXiv preprint arXiv:1909.11942*.

Lehečka, J., & Švec, J. (2021). Comparison of Czech Transformers on Text Classification Tasks. *arXiv preprint arXiv:2107.10042*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv preprint arXiv:1910.13461.*

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692.*

Loshchilov, I., & Hutter, F. (2017). Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101.*

Lund, K., & Burgess, C. (1996). Producing High-Dimensional Semantic Spaces from Lexical Co-Occurrence. *Behavior Research Methods, Instruments, & Computers*, *28*(2), 203–208.

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-Based Neural Machine Translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421.

Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., & Zamparelli, R. (2014). A SICK Cure for the Evaluation of Compositional Distributional Semantic Models. *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, 216–223. http://www.lrec-conf.org/proceedings/lrec2014/pdf/363_Paper.pdf

Micaelli, P., & Storkey, A. J. (2019). Zero-shot Knowledge Transfer via Adversarial Belief Matching. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 9551–9561). Curran Associates, Inc. http://papers.nips.cc/paper/9151-zero-shot-knowledge-transfer-via-adversarial-belief-matching.pdf

Michel, P., Levy, O., & Neubig, G. (2019). Are Sixteen Heads Really Better Than One? *arXiv preprint arXiv:1905.10650.*

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed Representations of Words and Phrases and Their Compositionality. *Advances in Neural Information Processing Systems*, 3111–3119.

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 746–751.

Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. *MIT Press.*

Mishra, A., & Marr, D. (2017). Apprentice: Using Knowledge Distillation Techniques to Improve Low-Precision Network Accuracy. *arXiv preprint arXiv:1711.05852.*

Mostafazadeh, N., Chambers, N., He, X., Parikh, D., Batra, D., Vanderwende, L., Kohli, P., & Allen, J. (2016). A Corpus and Evaluation Framework for Deeper Understanding of Commonsense Stories. https://doi.org/10.48550/arxiv.1604.01696

Mozetič, I., Grčar, M., & Smailović, J. (2016). Multilingual Twitter Sentiment Classification: The Role of Human Annotators. *PloS one*, *11*(5), e0155036.

Nivre, J., de Marneffe, M.-C., Ginter, F., Hajič, J., Manning, C. D., Pyysalo, S., Schuster, S., Tyers, F., & Zeman, D. (2020). Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection. *arXiv preprint arXiv:2004.10643.*

Pan, X., Zhang, B., May, J., Nothman, J., Knight, K., & Ji, H. (2017). Cross-lingual Name Tagging and Linking for 282 Languages. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1946–1958. https://doi.org/10.18653/v1/P17-1178

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543.

Pikuliak, M., Grivalský, Š., Konôpka, M., Blšták, M., Tamajka, M., Bachratý, V., Šimko, M., Balážik, P., Trnka, M., & Uhlárik, F. (2021). SlovakBERT: Slovak Masked Language Model. *arXiv preprint arXiv:2109.15254.*

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. https://openai.com/blog/language-unsupervised/

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. https://openai.com/blog/better-language-models/

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, *21*(140), 1–67.

Rahimi, A., Li, Y., & Cohn, T. (2019). Massively Multilingual Transfer for NER. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 151–164. https://www.aclweb.org/anthology/P19-1015

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. http://arxiv.org/abs/1908.10084

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, *323*(6088), 533–536.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. *arXiv preprint arXiv:1910.01108*.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., & Zhou, D. (2020). MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. *arXiv preprint arXiv:2004.02984*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., & Titov, I. (2019). Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. *arXiv preprint arXiv:1905.09418*.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2019). SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. *Advances in Neural Information Processing Systems*, *32*.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv preprint 1804.07461*.

Warstadt, A., Singh, A., & Bowman, S. R. (2018). Neural Network Acceptability Judgments. *arXiv preprint arXiv:1805.12471*.

Williams, A., Nangia, N., & Bowman, S. (2018). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 1112–1122. http://aclweb.org/anthology/N18-1101

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., . . . Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference* *on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

Wu, J., Ouyang, L., Ziegler, D. M., Stiennon, N., Lowe, R., Leike, J., & Christiano, P. (2021). Recursively Summarizing Books with Human Feedback. https://doi.org/10.48550/arxiv.2109.10862

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., & Raffel, C. (2021). MT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 483–498.

Zeman, D. (2017). Slovak Dependency Treebank in Universal Dependencies. *Journal of Linguistics/Jazykovedný Časopis*, *68*(2), 385–395.

Appendix A

Slovak UPOS tags

The following is a list of UPOS tags for the Slovak language. Examples are taken from Slovak Dependency Treebank[1].

ADJ: veľký, prvý, celý, nový, ďalší, druhý, dobrý, starý, posledný, slovenský
ADP: v, na, s, z, do, o, k, po, za, od
ADV: veľmi, potom, tu, tam, kde, tak, opäť, vtedy, ako, nikdy
AUX: byť, by, bývať
CCONJ: a, ale, aj, alebo, i, ani, či, však, lebo, no
DET: to, ktorý, jeho, svoj, ten, všetok, môj, jej, táto, tento
INTJ: mhm, haló, pozor, bum, aha, amen, klap, ach, pche, preboha
NOUN: rok, vláda, deň, človek, chvíľa, oko, ruka, mama, tvár, život
NUM: jeden, dva, tri, oba, ii, 1, štyri, 11, 2, mnoho
PART: aj, však, nie, len, už, a, až, iba, ani, ešte
PRON: sa, ja, on, ona, si, čo, ty, my, niečo, nič
PROPN: maja, chris, winston, aladin, vilko, Mauglí, jazmína, lori, bush, marga
PUNCT: ., „ “, !, ?, ",), (, :, „
SCONJ: že, keď, ako, aby, ak, kým, keby, akoby, čo, pretože
SYM: %, +, =, −
VERB: mať, byť, povedať, môcť, chcieť, ísť, vedieť, musieť, prísť, stať
X: o, tzv, sv, po, česko, km, c, r, the, č

Byte-pair encoding

Byte-pair encoding is a compression algorithm first introduced by (Gage, 1994). The algorithm replaces the most frequent pair of bytes with a new byte not present in the data. Replacing stops when no byte-pairs are repeated. A table of replacements is

[1]https://github.com/UniversalDependencies/UD_Slovak-SNK

kept, so the original bytes can be restored. An example can look like this:

$$aacbaacaa \rightarrow XcbXcX \rightarrow YbYX$$

The replacement table would in this example contain two entries: $X = aa$ and $Y = Xc$.

Appendix B

The electronic attachment contains all distilled models in folders named *experiment-[1-6]*. All models are saved in PyTorch format along with their configurations, vocabularies and tokenizers. Folder *masters-thesis* contains various scripts, including the training scripts. It also contains machine-translated datasets. Slovak STSBenchmark is in comma-separated values (CSV) format, while the Slovak BoolQ dataset is in JSON lines format. The C4 dataset is not included due to its large size. Specific versions of the libraries `sentence-transoformers` and `transformers` are also included in the electronic attachment.